ADVANCES IN SMART HEALTHCARE TECHNOLOGIES

# Intelligent Internet of Things for Smart Healthcare Systems

Edited by

**Durgesh Srivastava, Neha Sharma, Deepak Sinwar, Jabar H. Yousif, and Hari Prabhat Gupta**

CRC Press
Taylor & Francis Group

# 12 Deep Learning Approach for Classification of Alzheimer's Disease

*Abbas H. Hassin Alasadi and Faten Salim Hanoon*
University of Basrah

## CONTENTS



DOI: 10.1201/9781003326182-12 

## 12.1 INTRODUCTION

Alzheimer's disease (AD) is a progressive neurological disease, which worsens over time. Additionally, it is regarded as the most common cause of dementia [1]. The term AD was coined in 1901 when Dr. Alois Alzheimer discovered a disease in a patient named Auguste Deter, who died of the disease in 1906 [2] (see Figure 12.1). The disease that would bear his name was first described in the scientific literature in 1910. The symptoms of AD are a decrease in memory, an impairment of cognitive abilities, a lack of logical thinking and judgment, and difficulty in expression and comprehension. Symptoms result from damage or destruction of brain cells that impair thinking, learning, and memory. As the disease progresses gradually, nerve cells in other parts of the brain are damaged, resulting in a total loss of brain function. As life expectancy increases, the number of people suffering from AD is expected to rise dramatically in the future.

According to the World Alzheimer's Report [3] statistics, an estimated 50 million people had AD in 2015. By 2050, this number is expected to increase to 131.5 million people worldwide.

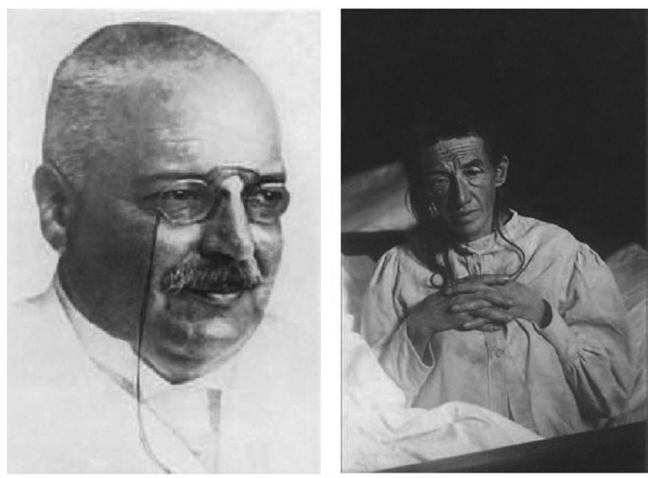

**FIGURE 12.1** Dr. Alois Alzheimer (left) and Auguste Deter (right). Deter was Dr. Alois Alzheimer's patient and the world's first patient diagnosed with AD [2].

## 12.2 DEEP LEARNING

Deep learning (DL) is a branch of machine learning and a subfield under the umbrella term "Artificial Intelligence" that involves building large neural network models capable of making accurate data-driven decisions [4]. The relationship between artificial intelligence, machine learning, and DL is shown in Figure 12.2. Dechter developed the concept of DL in 1986 [5]. In recent years, two key technological trends have driven the growth of DL. First, a huge amount of data is freely available. According to IDC, the global datasphere will reach 175 zettabytes by 2025 [6]. DL is particularly suited for complex data cases, and there are huge datasets to work with.

Second, one of the major factors for the widespread adoption of DL has been the innovation and advancement in the computational capabilities of parallel computing hardware, which has made it easy to parallelize the computations required for deep learning on these devices. DL has the following advantages over machine learning: The most important aspect of DL is that DL algorithms can automatically extract features from raw data, which helps to improve accuracy on a variety of problems [4]. In contrast, processing raw data and extracting features in traditional machine learning is limited and requires extensive expertise. It is considered a labor-intensive task that requires and consumes a significant amount of time and project budget. DL has a high learning capacity and also performs better based on the learning outcomes [7]. The depth of the model is the other aspect that distinguishes DL from other machine learning techniques. DL neural networks have multiple layers and a wide breadth of applications, so DL can solve highly complex problems [8]. Data dependency is a

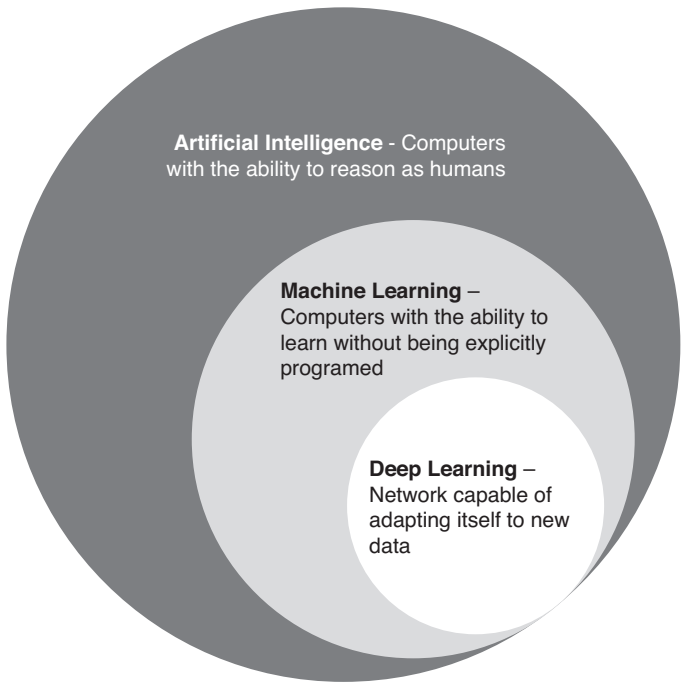


**FIGURE 12.2** The relationship between artificial intelligence, machine learning, and DL [4].

feature of DL. Relevant experiments have shown that the larger the amount of data available, the better the performance of DL.

Some tasks, such as image recognition, face recognition, and natural language processing, even outperformed human performance [9]. DL has excellent portability [10]. Due to the superior performance of DL, numerous frameworks such as MATLAB, TensorFlow, and Pytorch are now available for DL deployment. These frameworks are compatible with a variety of platforms, including Windows, Linux, and Mac OS. DL is now used by most online businesses and high-end consumer technologies. For example, Facebook uses DL to analyze text in online conversations. Google and Microsoft also use DL for machine translation and image search.

In addition, DL systems now run on all modern smartphones. It is used as a standard technology for speech recognition and face recognition on digital cameras. DL is also considered to be at the heart of self-driving cars. It is used for localization and mapping, motion planning and control, environmental awareness, and driver state tracking. In healthcare, DL is used to analyze medical imaging scans (X-rays, CT, and magnetic resonance imaging (MRI)) to make diagnoses. Convolutional neural networks (CNNs) have gained prominence in a variety of fields, including image processing and analysis, computer vision tasks, and medical imaging applications such as cerebral microbleed detection [11], automatic myocardial infarction detection [12], brain tumor segmentation [13], and COVID-19 detection [14]. In this chapter, CNN architecture is used in order to solve the four-stage classification problem of AD.

## 12.3 DL BUILDING BLOCK

Artificial neural networks (ANNs), also known as feedforward neural networks (FNNs) or multilayer perceptrons (MLPs), are the fundamental building blocks of numerous DL models that have achieved considerable success in processing high-dimensional imaging datasets [4]. ANNs simulate the neural network systems of the human brain. The human brain, which functions as the nerve system's command center, is composed of billions of neurons linked by approximately (1,014) synapses. Each neuron has three parts: a cell body, dendrites, and an axon, and is regarded as the brain's computing unit. The neuron takes input signals from its dendrites, processes them, and then transmits output signals via its axon. The biological neuron structure of the human brain is depicted in Figure 12.3a.

According to the mathematical model depicted in Figure 12.3b, signals (Si) that depend on the strength of synapses (Wi) interact multiplicatively (SiWi) with the dendrites of other neurons. This model can be used to demonstrate how synapse strengths (weights) can learn and control one neuron's influence on other neurons. Figure 12.3 illustrates the path taken by all signals from dendrites to the body cell, which accumulates all incoming signals. When the sum surpasses a specific threshold, neurons fire and send a signal spike via the axon.

In 1958, Rosenblatt created the perceptron [16], the world's first learning neural computer, to imitate human learning. The perceptron is an abstract model of a single

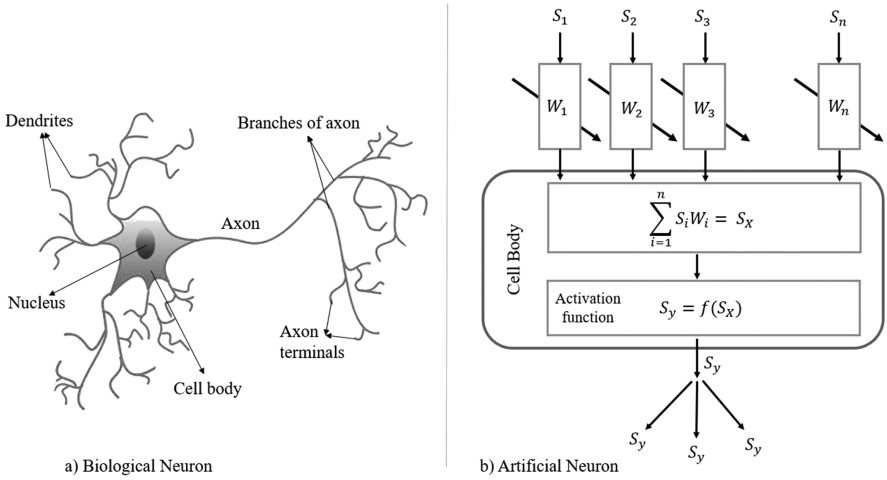


**FIGURE 12.3** (a) Biological neuron and (b) mathematical model for an artificial neuron [15].

neuron that has one output and several inputs. The output of the perceptron is the sum of all the weighted inputs xi plus the bias b as expressed in Equation (12.1).

$$f(x) = \varphi(b + \sum_{i=1}^{n} x_i \ .w_i) \ \ldots \tag{12.1}$$

The Heavyside step function is denoted by the symbol ($\varphi$), described by Equation (12.2).

$$\varphi(x) = \begin{cases} 1 & if \ x \geq 0 \\ 0 & else \end{cases} \quad \ldots \tag{12.2}$$

This network was used to perform binary classification. If the summation result is greater than or equal to zero, the network votes for the first class, and if the summation result is less than zero, the network votes for the opposite class.

Neural networks have been developed to improve the limited representational capabilities of the perceptron. They are made up of a series of arranged layers, each of which comprises a collection of perceptrons known as units or neurons. The input layer is the first layer, the output layer is the last, while layers that lie between the input and output layers are referred to as hidden layers. The nodes in one layer are connected to those in the next and previous layers. These connections are weighted edges, called weights. One of the most commonly used architectures is the multilayer perceptron, which has multiple hidden layers.

DL uses an architecture known as "deep neural networks. " Deep neural networks are a type of neural network that contains numerous hidden layers of neurons. For a neural network to be considered deep, at least two hidden layers are required [4]. However, most DL networks contain many more than two hidden layers

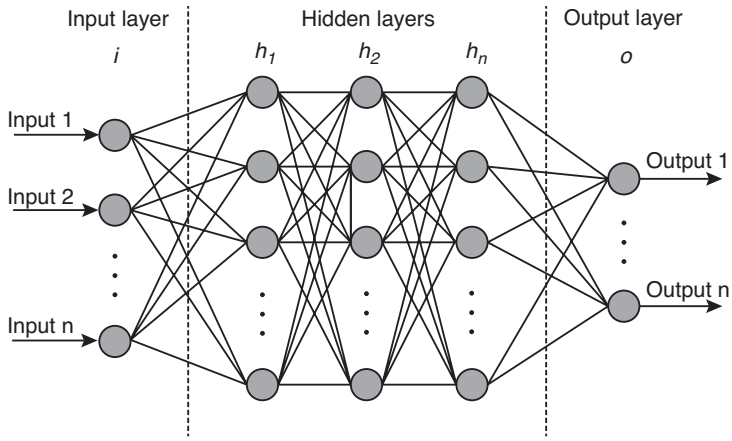


**FIGURE 12.4** ANN with multi-hidden layers [18].

(see Figure 12.4). The key point is that the depth of a network is defined by the number of hidden layers plus the output layer. However, deeper networks empirically outperform external networks with one hidden layer and have lower generalization errors. Nielson [17] attributes the superior performance of deeper networks to learning a complex hierarchy of concepts. In image classification, after the network has obtained all the required information from the input layer, each hidden layer detects a different set of features in an image, ranging from less detailed to more detailed. For example, the first hidden layer detects edges and lines, the second layer detects shapes, and the third layer detects specific image elements, such as a face or a wheel. The predictions of the network are made in the output layer. The predicted image classes are compared to the labels that were manually inserted by humans. If they are wrong, the network uses a technique called backpropagation (which will be discussed later) to adjust its learning process in order to provide more accurate estimates in subsequent iterations. After a sufficient amount of training, a network can make classifications automatically without the need for human intervention.

## 12.4 CONVOLUTIONAL NEURAL NETWORK (CNN)

A CNN is a special ANN that applies image processing directly to pixels without requiring any prior processing [19]. Yann LeCun proposed it for effective image recognition [4]. It is a part of DL technologies. CNNs are used in various applications, including image classification, segmentation, and pattern recognition [20,21]. Due to its autonomous nature, it has become an important tool for machine vision and artificial intelligence. A convolutional layer consists of units and an atypical neural network but with a different order and connection of units. The main differences between them and neural networks are the following:

1. The units are not arranged in one dimension but three dimensions. The colored image is responsible for the three-dimensional arrangement. A colored image typically has three channels (red, green, and blue), each of

which is represented by a two-dimensional matrix. As a result, ConvNet's input is a three-dimensional matrix. The output of a convolutional layer is a three-dimensional matrix with two-dimensional feature maps multiplied by the number of filters in that layer. Each filter generates one feature map.

2. Weight sharing: The use of the same weights for various output units is called weight sharing, which results in ConvNet having one property, feature invariance to translation. This means that the feature is present on the entire input.
3. Local connectivity: Local connectivity is a term that relates to the concept of each neuron being connected to a portion of the input image, as contrasted to a neural network in which all neurons are connected to the full input image. This contributes to the reduction of the total number of parameters in the system and increases the computation efficiency.

### 12.4.1 Basic Building Blocks of CNNs

In this section, the basic layers of the CNN will be explained in detail.

#### 12.4.1.1 Convolutional Layer

The convolutional layer is the building block of the CNN algorithm. It is responsible for extracting the essential and useful features from the input images using a set of trainable filters that form a feature map [22]. Convolution is a mathematical technique in which a filter is applied to an n-dimensional field (image). The filter also consists of a set of numbers called weights or parameters. The values are multiplied by the original pixel values of the image as the filter slides over or convolves the input image [23]. In other words, it multiplies element by element. All of these multiplications are summed. When the filter slides over the entire image, the result is a filtered image (called the feature map). In one layer, many filters are applied to the initial image, and each filter represents a set of weights that were learned during the training process. Equation (12.3) expresses the convolution operation.

$$m_j^f \to (s,\ t) = \sum_z \sum_{x,y} r_z\,(x,y).\ e_j^f.(w,\ h)\ \ldots \tag{12.3}$$

where $r_z\,(x,y)$ represents an element of the input image tensor $R_Z$, $x$ is the $x^{th}$ coordinate under consideration of an image, $y$ is the $y^{th}$ coordinate under consideration of an image, $z$ is the index of the channel, $e_j^f$ (w, h) is the index of the $f^{th}$ convolutional filter $f_j$ of the $j^{th}$ layer, $J$ is the total number of layers, $j$ is the layer number, $F$ is the total number of filters of the $j^{th}$ layer, $f_j$ is the filter number of the $j^{th}$ layer, $w$ is the $w^{th}$ row under consideration, $h$ is the $h^{th}$ column under consideration, $S$ is the total number of rows of the feature matrix, $T$ is the total number of columns of the feature matrix, $s$ is the $s^{th}$ row under consideration, $t$ is the $t^{th}$ column under consideration, and $m_j^f$ $(s, t)$ is an element of the feature map shown in the notion:

$$\boldsymbol{M_j^f} = \left[\boldsymbol{m_j^f}\ (1,1),\ldots.,\boldsymbol{m_j^f}\,(\boldsymbol{s},\boldsymbol{t}),\ldots\boldsymbol{m_j^f}\,(\boldsymbol{S},\boldsymbol{T})\right]$$

where $M_j^f$ is the input feature matrix for the $j^{th}$ layer and the $f^{th}$ neuron.

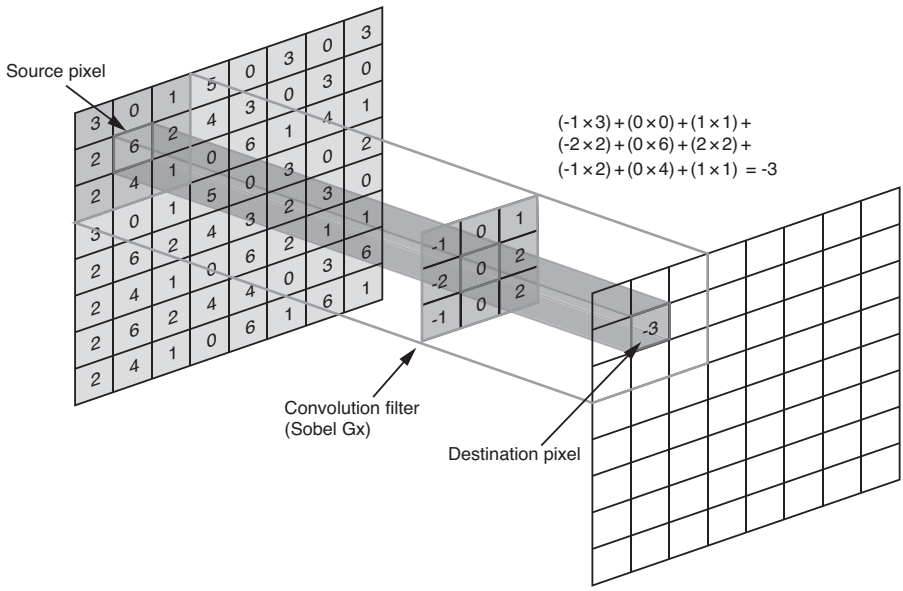


**FIGURE 12.5** Visualization of the convolution process [24].

As shown in Figure 12.5, the filter matrix whose size is (3×3) was used with the matrix in a region in the input image whose size is (3×3) to perform a dot product multiplication. Then, the resulting matrix elements are added, and the sum yields a single numerical value (target pixel) on the feature map. This process is repeated by moving the matrix of the filter over the input matrix to complete the feature map by multiplying the dot product by each remaining combination of 3×3 sized areas. A set of filters are applied to an input image and the resultant feature maps are combined to provide the final output of a single convolutional layer.

Convolutional layers have two other key concepts: strides and padding [24]. A stride is the number of pixels that a kernel or filter moves across the input matrix. The default value for strides is 1, but occasionally a stride greater than 1 is used to minimize feature maps. At the same time, padding is used when the filter does not fit the input matrix. Padding is classified into two types: valid padding, which discards the input matrix's edge pixels, and null or equal padding, which adds zeros to the edges to make the filter fit the input matrix. In addition, two hyperparameters are important for convolution operations: the first is the kernel size, which is usually 3×3 but can sometimes be 5×5 or 7×7, and the second is the number of kernels, which determines the depth of the output feature maps [25].

### 12.4.1.2 Pooling Layer

The pooling layer (also called the down-sampling layer) is responsible for reducing the spatial size of the convolved feature [26]. Dimensionality reduction reduces the computational power required to process the data. As a result, it reduces the number of parameters and the risk of over-fitting the data. It also helps in extracting the most

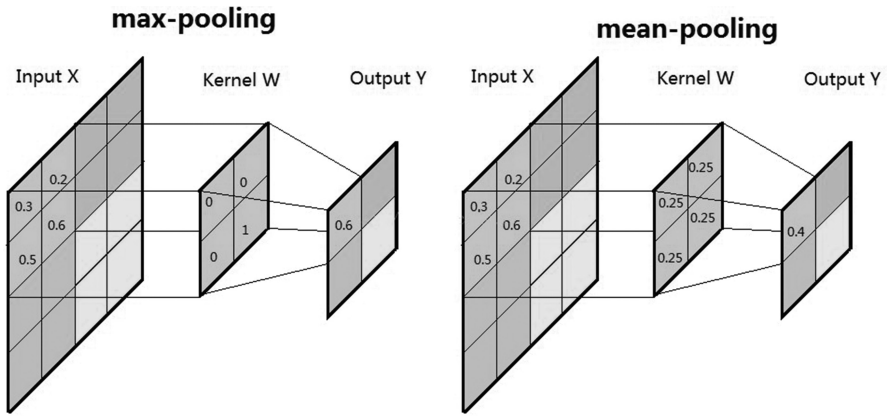


**FIGURE 12.6** Pooling operation (max pooling, average pooling) [24].

important and useful features. The fact that there are no learnable parameters in any of the pooling layers should be noted. Pooling operations, like convolution operations, use hyperparameters like filter size, stride, and padding [25]. Equation (12.4) illustrates the pooling operation, where $P_j^f$ represents the feature map after pooling the $j^{th}$ layer for the $f^{th}$ Input feature map $M_j^f$, $g_p$ (.) determines the type of pooling operation.

$$P_j^f = g_p(M_j^f)\ldots \tag{12.4}$$

Max pooling is the most widely used pooling method, where patches are taken from the input feature maps, the highest value is output, and the rest is discarded (see Figure 12.6). In practice, max pooling is often used with a filter size of $2\times2$ and a stride of 2. Average pooling is the other type of pooling operation. Average pooling returns the average value of the pixels of the image covered by the kernel. Usually, this operation is performed only once before the fully connected layers.

#### 12.4.1.3 Activation Layer

Typically, after each convolutional layer, an activation layer is applied. It gives nonlinear characteristics to a system that has recently completed a linear calculation in a convolutional layer. The activation function serves as a decision-making mechanism and facilitates the learning of complex patterns. Utilizing the appropriate activation function helps accelerate the learning process. The activation function for a convolved feature map is defined by Equation (12.5).

$$K_j^f = g_a\left(M_j^f\right)\ldots \tag{12.5}$$

According to the formula of Equation (12.5), a convolution's output $\left(M_j^f\right)$ is given to an activation function $g_a$ (.), which adds nonlinearity and returns a transformed output $K_j^f$ for the $j^{th}$ layer. A variety of activation functions are used in the literature to inculcate nonlinear combinations of features, such as sigmoid, tanh, maxout, SWISH,

rectified linear unit (ReLU), and versions of ReLU, such as leaky ReLU, ELU, and PReLU. ReLU and its variants, on the other hand, are recommended because they help overcome the vanishing gradient problem [27].

#### 12.4.1.4 Batch Normalization Layer

These layers are typically placed after activation layers, yielding normalized activation maps by subtracting the mean and dividing it by the standard deviation for each training batch. The network is forced to periodically change its activations to zero mean and unit standard deviation as the training batch passes through these layers by including batch normalization layers. This acts as a regularizer for the network, accelerates training, and reduces the network's reliance on careful parameter initialization [28]. Equation (12.6) illustrates batch normalization for a transformed feature map $F_l^k$.

$$N_j^f = \frac{M_j^f - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \quad \ldots \tag{12.6}$$

where $M_j^f$ is the input feature map, and $\mu_B$ and $\sigma_B^2$ represent the mean and variance of a feature map for a mini batch, respectively. $N_j^f$ is the normalized feature map.

#### 12.4.1.5 Dropout Layer

The dropout layer is typically used to control over-fitting to prevent it from occurring [29,30]. During the forward pass, it discards a random activation parameter set by setting it to zero to ensure that the neural network will not affect the training samples overmatching, thereby alleviating over-fitting issues.

#### 12.4.1.6 Fully Connected Layer

The fully connected layer is typically found at the network's end and is utilized for classification purposes. It is a global operation, unlike pooling and convolution. It receives inputs from the feature extraction layers and analyses the outputs of all the previous layers at a global level. As a result, a nonlinear combination of selected features is created, which is used to classify the data.

The network uses a fully connected layer to map higher level activation mappings to the output layer classification and construct an n-dimensional vector, where n denotes the number of classifications in the output layer [31]. This *n*-dimensional vector represents the probability of the recognized image in N classifications. Figure 12.7 illustrates the fundamental building blocks of a typical CNN.

### 12.4.2 Training CNN

A CNN is trained by finding kernels in convolutional layers and weights in fully connected layers that reduce the differences between output predictions and predefined ground-truth labels in the training dataset. A backpropagation algorithm is a popular approach to neural network training that relies heavily on the loss function, and gradient descent optimization algorithm. A loss function computes the performance of a model under certain kernels and weights using forward propagation on a training

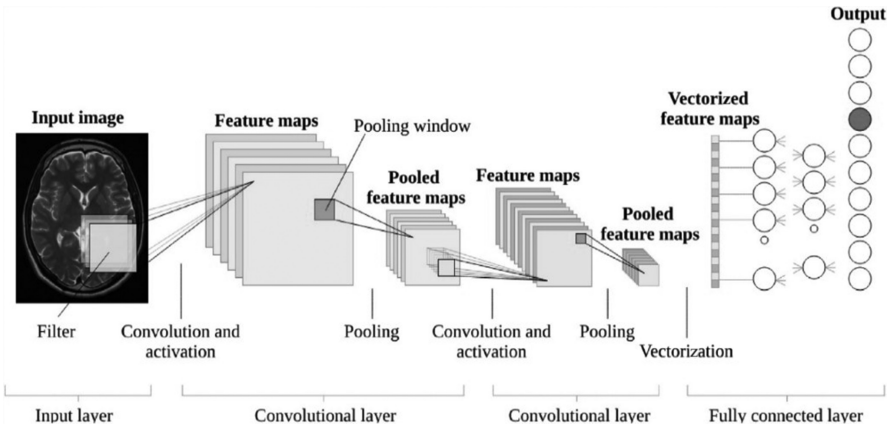


**FIGURE 12.7** The fundamental building blocks of a typical CNN [28].

dataset. Learning parameters such as kernels and weights are updated according to the loss value using a backpropagation algorithm and gradient descent.

Fully training a new CNN from scratch is not without challenges. First, a CNN requires large amounts of labeled data for the training process, which can be difficult to obtain, especially in medical imaging. In addition, training a CNN requires the use of many computational and storage resources. Otherwise, without these resources, the training process would take a very long time. Tuning hyperparameters is time-consuming and complicated and may lead to over-fitting or under-fitting, resulting in poor model performance. Researchers have developed a promising alternative method, called transfer learning, to overcome these obstacles.

Transfer learning involves improving a new task by transferring knowledge from a previously learned task [32].

### 12.4.3 Basic CNN Architecture

In this section, the basic CNN architectures are explained.

#### 12.4.3.1 LeNet-5

The LeNet-5 is the first CNN architecture proposed by LeCun et al. [33] in 1998 for classifying handwritten digits. The LeNet-5 consists of five trainable layers, of which three are convolutional, and two are fully connected. The first two convolutional layers are each followed by a max-pooling layer and two fully connected layers follow the last convolutional layer. The final layer of these fully connected layers serves as a classifier that can classify ten digits. The architecture of LeNet-5 is shown in Figure 12.8.

#### 12.4.3.2 AlexNet

Krizhevky et al. [34] created the first large CNN model called AlexNet in 2012, which is based on LeNet and is employed to classify ImageNet data. It has eight

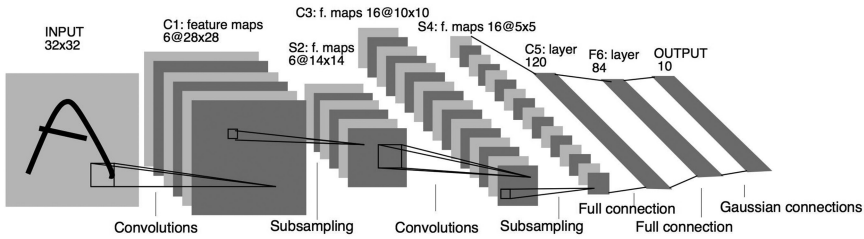


**FIGURE 12.8** The architecture of LeNet-5 [36].

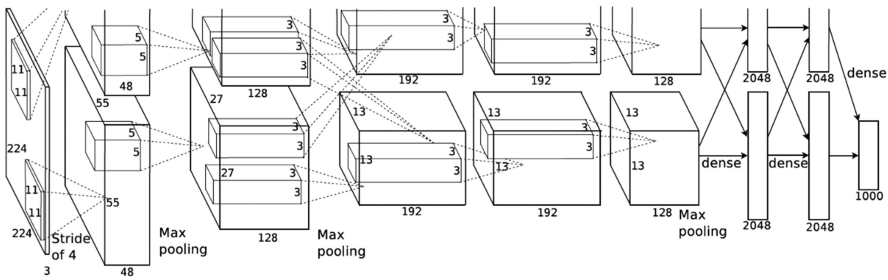


**FIGURE 12.9** The architecture of AlexNet [36].

layers for learning, the first five of which are convolutional and the last three of which are fully connected [35]. Because it was developed for ImageNet data, the final output layer classifies the input images with 1,000 units into one of the ImageNet dataset's 1,000 classes. The architecture of AlexNet is shown in Figure 12.9.

#### 12.4.3.3 ZFNet

Zeiler and Fergus presented ZFNet [37] at ECCV-2014. It has a similar design to AlexNet, with the exception that the first convolutional layer uses $7 \times 7$ filters with Stride 2. When it comes to the first convolutional layer of AlexNet, Krizhevky et al. employed an $11 \times 11$ filter with Stride 4. As a result, ZFNet exceeds AlexNet in terms of efficiency, earning it the title of ILSVRC-2013 winner. Figure 12.10 depicts the ZFNet architecture.

#### 12.4.3.4 VGG

VGG is one of the most widely used CNN architectures, introduced in 2014 by Simonyan and Zisserman. VGGNet's popularity arises from its simple architecture and the use of small-scale filters for convolutional processes [36]. This network illustrates that, when used, a stack of filters of size $3 \times 3$ in convolution operation has the same effective effect as a stack of large-sized filters. For example, two layers of filter with a size of $3 \times 3$ have the same effect as a filter with a size of $5 \times 5$ in convolution operation. Three layers of filters with a size of $3 \times 3$ have the same effect as filters with a size of $7 \times 7$. The important thing is that when filters of small size are used, the

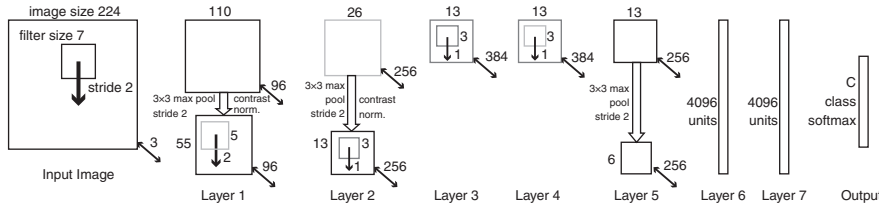


**FIGURE 12.10** The architecture of ZFNet [36].

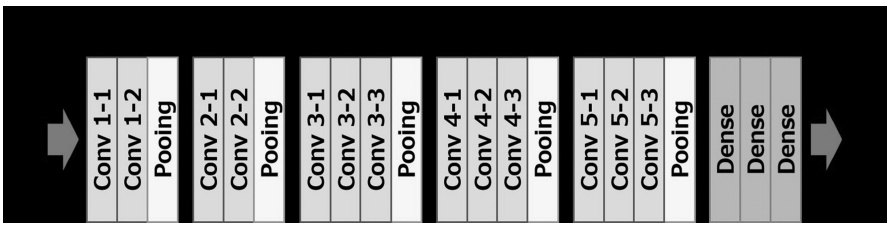


**FIGURE 12.11** The architecture of VGG [36].

number of parameters in the network is reduced. The architecture of VGG is shown in Figure 12.11.

#### 12.4.3.5 GoogLeNet

GoogLeNet was proposed by Szegedy et al. in 2014. In contrast to the traditional CNN models previously addressed, GoogLeNet [36] employs network branches rather than a single-line sequential architecture. The GoogLeNet has 22 learnable layers and is built using the Inception Module, which represents the fundamental building block for this network. This module's processing occurs in parallel across the network. Each module is composed of convolution layers with filters of the following sizes: $1\times1$, $3\times3$, and $5\times5$, which operate in a parallel way, resulting in combined feature maps with extremely high dimensions. To address the issue of the generated feature maps having high dimensions, they used the inception module to reduce the dimensions (as illustrated in Figure 12.12).

#### 12.4.3.6 ResNet

Backpropagation over a deep CNN (a CNN with a large number of layers, e.g., 1,000) requires the computation of loss gradients (errors) concerning the corresponding weights in the neurons of each layer that update these weights. This task uses the derivative operation, which causes the gradients to become smaller and smaller. For this reason, the neurons in the earlier layer receive minimal gradients (sometimes the gradients become almost zero), which causes the weights in the previous layer to receive only minor updates, and learning for these layers becomes slow and inefficient. This is known as the "vanishing gradient problem" [36].

The residual neural network, known as ResNet, was proposed in 2016 by He et al. [38] to solve this problem. ResNet ranked first in the 2015 ILSVRC classification

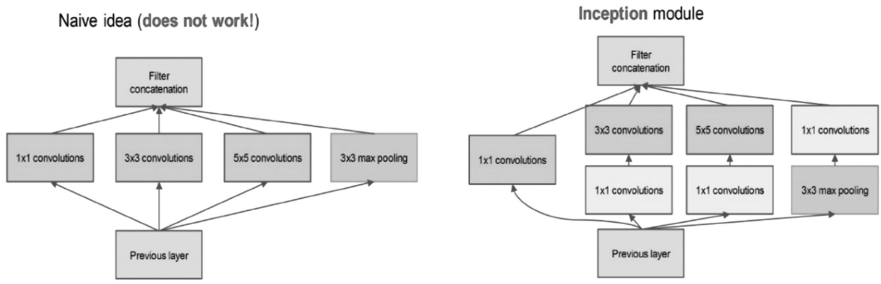


**FIGURE 12.12** On the right, simple inception module, with dimensionality reduction on the left inception module [36].

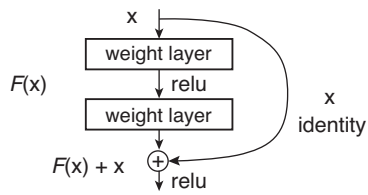


**FIGURE 12.13** Structure of residual block [36].

competition with an error rate of 3.57. ResNet inventively uses shortcuts known as "skip connections," i.e., direct connections between two non-consecutive layers [39,40]. Adding the input x to the output after a few convolutional layers avoids the vanishing gradient problem. Residual blocks are designed to fit a residual mapping $F(x)$ rather than the desired underlying mapping $H(x)$ to assist in the optimization of deeper models, and entire ResNet architectures are built by stacking residual blocks. Figure 12.13 illustrates the concept of a residual block. If the input is $x$, the output of the convolutional layer is $F(x)$, which is added to $x$ as a mapping input, and the resulting output $H(x) = F(x) + x$ is passed to the next layer. This is much simpler than fitting an identity map through a collection of nonlinear layers, and the network does not need to include additional parameters and computations. At the same time, the training speed and effectiveness of the model can be significantly increased as the number of layers increases. This residual block structure can effectively solve the gradient disappearance problem in deep networks [38]. There are two types of residual blocks in ResNet. While the first type is suitable for training shallow networks, the second type (bottleneck) is recommended for more than 50 layers. Moreover, both types have a similar time complexity.

## 12.5 PROPOSED FRAMEWORK

Accuracy in medical diagnosis is more important than anything else, even more important than speed of diagnosis. After all, the wrong diagnosis of an ordinary person as a patient causes severe consequences and psychological pressure, as well as

diagnosing a patient, as normal, leads to the development of the disease because the wrong diagnosis, in this case, will delay treatment. Therefore, building an automated medical diagnostic system must be of a high level of accuracy.

This section presents the general architecture of the proposed framework. The proposed framework consists of the following stages: data collection, data preparation, model selection, and training stage to build the model that helps in diagnosis, validation, and evaluation. Each stage is independent of the other and is responsible for implementing a specific function. At the same time, these stages can communicate with each other since the result of one stage will be the input to the following stage. Figure 12.14 describes the proposed framework.

### 12.5.1 Data Collection Stage

Data collection is the first step in the machine learning pipeline for training the selected model. The accuracy of machine learning systems' predictions is only good when the data used to train them is good. Therefore, the first stage in the framework of the proposed work in this chapter is to collect data and obtain it from data sources related to this work in order to solve the research problem, test the hypothesis, and evaluate the results.

In this chapter, the Alzheimer's brain MRI dataset was obtained from the open access of the Kaggle website. The dataset contains 6,400 images with a size of $176 \times 208$ pixels. It has four classes (NonDemented, MildDemented, ModerateDemented, VeryMildDemented) with a non-uniform distribution of the images per class.

### 12.5.2 Data Preparation Stage

This stage contains four steps for preparing the selected data:

#### 12.5.2.1 Convert to RGB

In this step, the images are converted to RGB because the network used in our work is pre-trained on color images.

#### 12.5.2.2 Resize

In this step, the size of images is changed to different sizes according to the selected experiment, for example, $(50 \times 50)$, $(75 \times 75)$, and $(125 \times 125)$. This step decreases the time for training the neural network by lowering the number of pixels in an image because more pixels in an image lead to an increase in the model's complexity. Another reason is that try the training of the network with different sizes of images.

#### 12.5.2.3 Augmentation

Because of the non-uniform distribution of images in each class, new training examples are generated using one of the data augmentation techniques only on the training set to improve deep neural network generalization capabilities and prevent over-fitting. Horizontal flipping is the augmentation technique used in this step. This technique works by shifting all of the pixels in an image in the horizontal direction, or in other words, by reversing the entire rows and columns of image pixels horizontally.

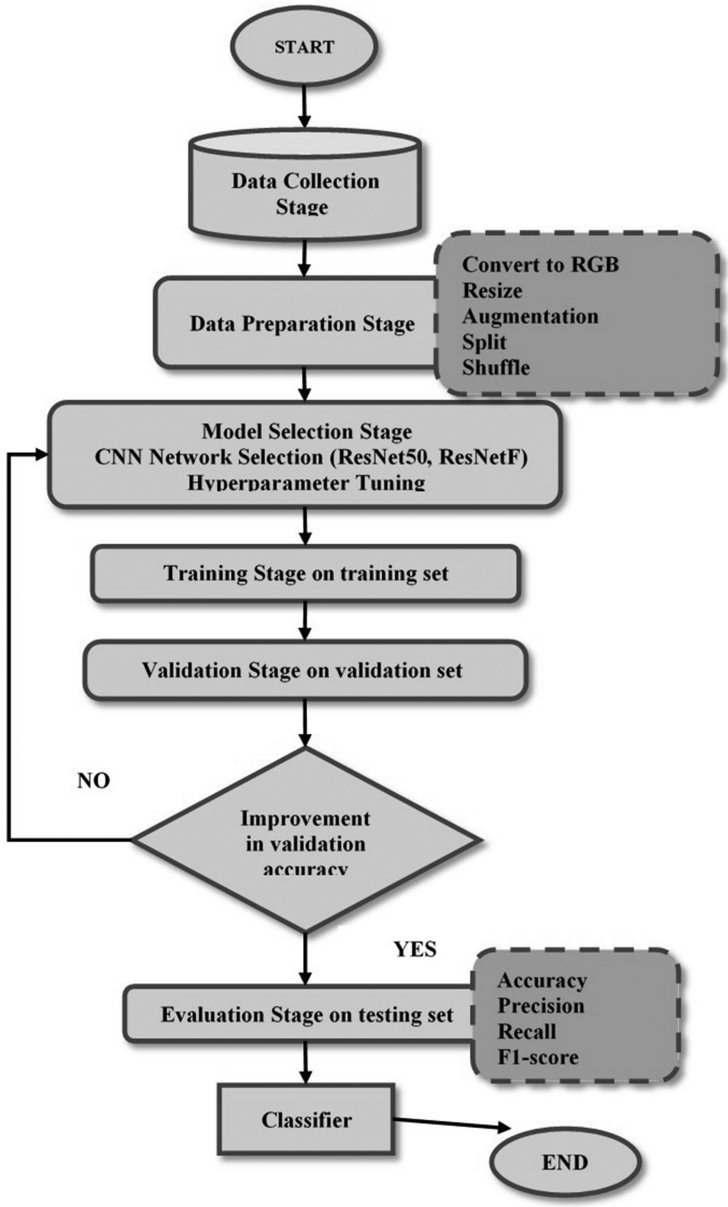


**FIGURE 12.14** Flowchart of the proposed framework for early diagnosis of AD.

#### 12.5.2.4 Splitting

The dataset is separated into two independent sets; the training set and the testing set. The training set is then partitioned into a validation set and a new training set. The proposed model is trained using the new training set. In contrast, the validation set periodically evaluates the model's performance during the training phase to avoid over-fitting problems. The testing set is later used to evaluate how well the model generalizes to unseen data.

#### 12.5.2.5 Shuffling

Shuffling is the last step in data preparation. This step shuffles the training data after each epoch to pass different inputs to the neurons in each epoch. This procedure prevents the model from learning the order of training samples and thus prevents bias. This step eventually helps the training to converge quickly so that the network can provide better generalizations.

By way of the validation set and testing set, no shuffling process was performed. During the validation phase and testing phase, there is no updating process for the model's parameters. During the validation and testing phases only, accuracy and loss are calculated. Their calculation method is not sensitive to the order of samples, so shuffling does not affect the testing and validation data.

### 12.5.3 Model Selection Stage

In this stage, the structure of a model is chosen. The algorithm and hyperparameters for the training stage are preliminarily determined.

#### 12.5.3.1 Algorithm Selection

In this chapter, two CNNs are studied to build a model for the early diagnosis of AD. The structure of each network and its components are explained in detail.

##### *12.5.3.1.1 ResNet-50 Architecture*

ResNet-50 is a residual DL network that deals with vanishing gradients in deep CNNs. During backpropagation, jump connections are used to jump across three layers.

The residual block in ResNet-50 always consists of $1 \times 1$, $3 \times 3$, and $1 \times 1$ convolutional layers stacked on top of each other. Figure 12.15 illustrates the concept of a residual block in ResNet-50.

The architecture of resnet50 consists of the following components:

1. A convolutional layer contains 64 different kernels. Each kernel has a size of $7 \times 7$ and a step size of two, followed by a max-pooling layer of the same size as the kernel step.
2. The first residual block contains a convolutional layer that has 64 kernels and each kernel is in the size of $1 \times 1$. Another convolutional layer follows this layer with $3 \times 3$ and 64 kernels. The final layer is also a convolutional layer that has 256 kernels and each kernel has a size of $1 \times 1$. This block is repeated three times, giving this step nine layers.
3. The second residual block contains a convolutional layer which has 128 kernels and each kernel is in the size of $1 \times 1$. Another convolutional layer

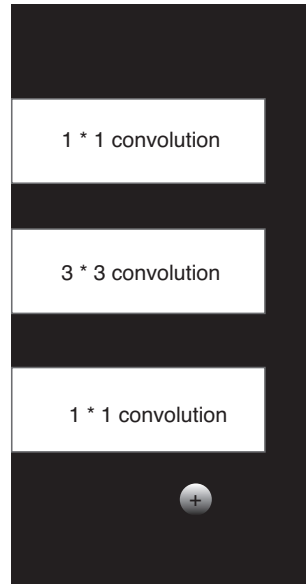


**FIGURE 12.15** Structure of bottleneck block in ResNet-50.

follows this layer with $3 \times 3$ and 128 kernels. The final layer is also a convolutional layer which has 512 kernels and each kernel has a size of $1 \times 1$. This block is repeated four times, giving this step 12 layers.

4. The third residual block contains a convolutional layer which has 256 kernels and each kernel is in the size of $1 \times 1$. Another convolutional layer follows this layer with $3 \times 3$ and 256 kernels. The final layer is also a convolutional layer which has 1024 kernels and each kernel has a size of $1 \times 1$. This block is repeated six times, giving this step 18 layers.
5. The fourth residual block contains a convolutional layer which has 512 kernels and each kernel is in the size of $1 \times 1$. Another convolutional layer follows this layer with $3 \times 3$ and 512 kernels. The final layer is also a convolutional layer which has 2048 kernels and each kernel has a size of $1 \times 1$. This block is repeated six times, giving this step nine layers.
6. This is followed by an average pooling layer, followed by a fully connected layer with 1,000 nodes, replacing this with four nodes according to the number of disease stages. The final layer is a softmax function, resulting in one layer.

In each residual block, after the convolutional layer come the batch normalization layer and then the activation function layer that uses ReLU, except for the last convolutional layer, which is only followed by batch normalization.

#### *12.5.3.1.2 ResNetF Architecture*

ResNetF is a modified residual neural network that is proposed based on ResNet50. The modification of RESNET50 is made as follows:

1. A convolution layer that has 64 different kernels. The size of each kernel is $7 \times 7$ and each has a step size of 2, followed by a max-pooling layer with the same step size as the kernel.
2. The number of convolution layers is increased by repeating the first residual block three times, the second residual block six times, the third residual block seven times, and the fourth residual block three times. The expanding number of convolution layers leads to the extraction of richer and more diverse features from the different layers. Because the more deeply embedded the network is, the more abstract the features that are extracted. Thus, the network's ability to extract features improves, and its effectiveness in diagnosing AD is increased. As a result, 58 convolution layers are structured in the proposed network.
3. Each residual block has a different number of kernels and comprises three convolution layers with different kernel sizes. Following the first and second layers are the batch normalization and activation function layers. As for the final layer, it is followed only by the batch normalization layer. Figure 12.16 illustrates layers of the residual block.
4. The average pooling layer is used after the last residual block. Then, to ensure that over-fitting is effectively avoided when this network is used, a dropout layer is added before the fully connected layer. The dropout ratio is set to 50%. In the end of the network, a fully connected layer was added that contains four nodes, based on the number of disease stages, and it concludes with a softmax function. See Figure 12.17.
5. In ResNet50, the ReLU is commonly used as an activation function. Basically, on CNN, ReLU takes the negative parts of its input and drops them to zero, and retains the positive parts. However, these negative inputs may contain useful feature information that could aid in the development of high-level discriminative features [41]. If a neuron's output is 0, its gradient

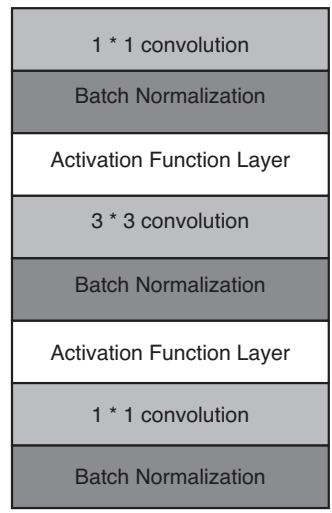


**FIGURE 12.16** Layers of the residual block.

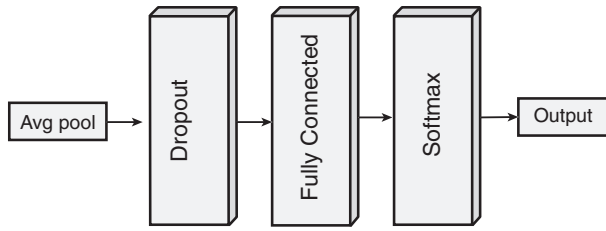


**FIGURE 12.17** Last layers in ResNetF architecture.

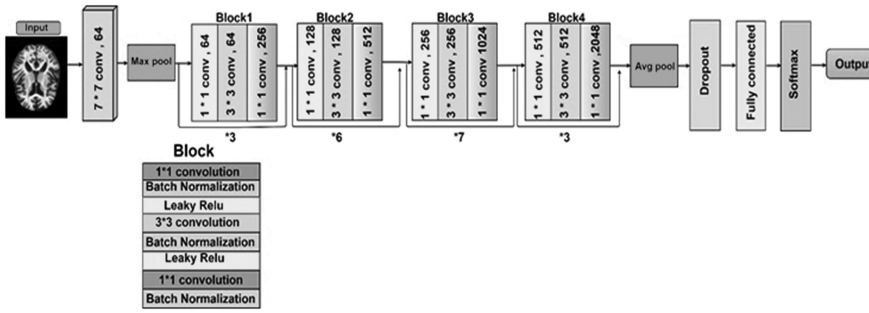


**FIGURE 12.18** Modified residual neural network (ResNetF) structure.

will never update its weight, resulting in the neuron never being activated. When the network contains a high number of inactive neurons, the convergence of the model becomes extremely difficult. This may prevent the network from learning and result in underperformance. Accordingly, this is referred to as the dying ReLU problem [42].

6. For this reason, ReLU is replaced with leaky ReLU (LReLU) in order to prevent any potential loss of input information. LReLU has added an alpha parameter to the semi-axis of ReLU, resulting in a slight gradient but not zero. Nodes that were previously inactive with ReLU will now be weight-adjusted. Figure 12.18 shows the ResNetF network.

#### 12.5.3.2 Hyperparameter Tuning

Hyperparameters are parameters whose values are used to control and regulate the learning process. The adjustment of hyperparameters has a significant impact on the accuracy of the trained model. Therefore, an optimal set of hyperparameters must be selected for the learning algorithm before it is trained. This process is called hyperparameter tuning. Experiments have been conducted to tune the hyperparameters, which are shown in Table 12.1.

### 12.5.4 Training Stage

In this stage, five experiments have been conducted to train the ResNetF architecture from scratch and perform transfer learning with the ResNet-50 architecture using the

**TABLE 12.1**
**A Set of Experiments to Tune the Hyperparameters**

| Hyperparameter | EXP1 | EXP2 | EXP3 | EXP4 | EXP5 |
|---|---|---|---|---|---|
| Size of image | (50×50) | (50 ×,50) | (75×75) | (125×125) | (125×125) |
| Size training sample | 65% | 70% | 70% | 80% | 80% |
| Size validation sample | 35% | 30% | 30% | 20% | 20% |
| No. of training samples | 3,328 | 3,584 | 3,584 | 4,096 | 4,096 |
| No. of validation samples | 1,793 | 1,537 | 1,537 | 1,025 | 1,025 |
| Batch size | 10 | 20 | 20 | 40 | 40 |
| Length training batches | 333 | 180 | 180 | 103 | 103 |
| Length valid batches | 180 | 77 | 77 | 26 | 26 |
| No. of epochs | 30 | 50 | 50 | 100 | 100 |
| Learning rate | 0.05 | 0.03 | 0.03 | 0.0003 | 0.00001 |

hyperparameters. In particular, two architectures are used in the experiments during the training phase. First, a pre-trained network (ResNet-50) is used to initialize the weights. Transfer learning was performed using a technique called off-the-shelf (OTS) transfer learning. In this approach, the last dense layer of the original network was replaced by a new dense layer corresponding to the number of classes in our task. In the standard approach, all layers of the ResNet-50 network except the last layer (classifier) are used for feature extraction. The weights of the last layer were adjusted to meet the requirements of our task. Second, a modified version of ResNet50 (ResNetF) is used to perform the training from scratch by randomly initializing the network parameters.

The training dataset was fed into our training networks during each epoch in batches form. Cross-entropy is used as the loss function. It is used when there are more than two classes in a classification problem. Adam is used as the optimizer, and the backpropagation algorithm was used to train the network. In the previous chapter, Adam and the backpropagation algorithm are described in detail.

### 12.5.5 Validation Stage

In this stage, a validation set is used, which is a sample of data that does not participate in the model training process. This data is used for the purpose of measuring the performance of the model after each epoch during the training phase to adjust the parameters of the model. The validation accuracy and loss compute the validation set to assess the model's performance after each epoch. This can determine whether the model suffers from bias (under-fitting) or variance (over-fitting). If the model suffers from over-fitting or under-fitting, the hyperparameters are retuned and the network is trained again. However, when the validation accuracy improves, the validation loss decreases. In this case, the training process continues until an optimal model is obtained. Then, the model with its parameters is saved as an H5 file to preserve the learned features after reaching the desired accuracy level. After that, the model is imported into the evaluation stage for final testing.

## 12.6 EVALUATION METRICS

Classification model performance is evaluated using unseen data (testing data) by the following metrics [19,43].

### 12.6.1 Confusion Matrix

The confusion matrix is a two-dimensional matrix that can visualize the performance of the classification model, also known as the error matrix. By default, the confusion matrix is designed for binary-class classification. However, it can also be extended to classify multiple classes. An example of a confusion matrix for binary-class classification is shown in Table 12.2.

The row labels positive and negative refer to the model's predictions. In contrast, the column labels positive and negative refer to the dataset's ground-truth labels. As for the entries inside the confusion matrix, they represent the following:

True positive (**TP**): the number of instances correctly categorized as positive by the model.

True negative (**TN**): the number of instances correctly categorized as negative by the model.

False positive (**FP**): the number of negative instances incorrectly categorized as positive by the model.

False negative (**FN**): the number of positive instances incorrectly categorized as negative by the model.

### 12.6.2 Accuracy

Model accuracy is defined as the ratio of correctly classified samples to the total number of samples. It is denoted mathematically by Equation (12.7).

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \ldots \tag{12.7}$$

### 12.6.3 Recall

The recall is the ratio of truly positive predicted instances to all positive instances observed in the ground data. It indicates the classification performance of positively labeled instances. It is also known as sensitivity or true positive rate (TPR). It is denoted mathematically by Equation (12.8).

**TABLE 12.2**
**Example Confusion Matrix for Binary-Class Classification**

| | | Actual Class | |
|---|---|---|---|
| **Confusion Matrix** | | **Positive (p)** | **Negative (N)** |
| Predicted class | Positive (p) | True positive (TP) | False positive (FP) |
| | Negative (N) | False negative (FN) | True negative (TN) |

$$\text{Recall} = \frac{TP}{TP + FN} \ldots \tag{12.8}$$

### 12.6.4 Precision

Precision is the ratio of correctly predicted truly positive instances among all instances classified as positive. It is denoted mathematically by Equation (12.9).

$$\text{Precision} = \frac{TP}{TP + FP} \ldots \tag{12.9}$$

### 12.6.5 F1-Score

Precision and recall frequently have an inverse relationship, increasing one at the expense of decreasing the other. Thus, a metric that balances these two metrics is needed. This is why the F1 score was created. It is known as the harmonic mean of precision and recall. It is denoted mathematically by Equation (12.10).

$$F1 - \text{score} = 2 \times \frac{\textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}} \ldots \tag{12.10}$$

## 12.7 EXPERIMENTAL RESULTS

Table 12.3 offerings the final result of the experiment ResNetF, while the accuracy of training and validation across epochs and the training loss and validation loss across epochs are shown in Figure 12.19. Table 12.4 shows the performance measures of ResNetF. Moreover, the confusion matrix of the current experiment is shown in Table 12.5.

## 12.8 CONCLUSION

AD is a degenerative neurological illness that worsens with age and leads to severe thinking, memory, and behavioral impairment. It is also considered the most common cause of dementia. Early diagnosis of AD is crucial because early intervention in AD slows the progression of the disease, accelerates the development of treatment options in the future, and reduces the financial burden on patients' families. Therefore, the task of early diagnosis of AD is the focus of many researchers who have built many CAD systems to diagnose AD.

**TABLE 12.3**
**Results of ResNetF**

| **Training Accuracy** | **99%** |
|---|---|
| Validation accuracy | 97% |
| Train loss | 0.04 |
| Validation loss | 0.08 |

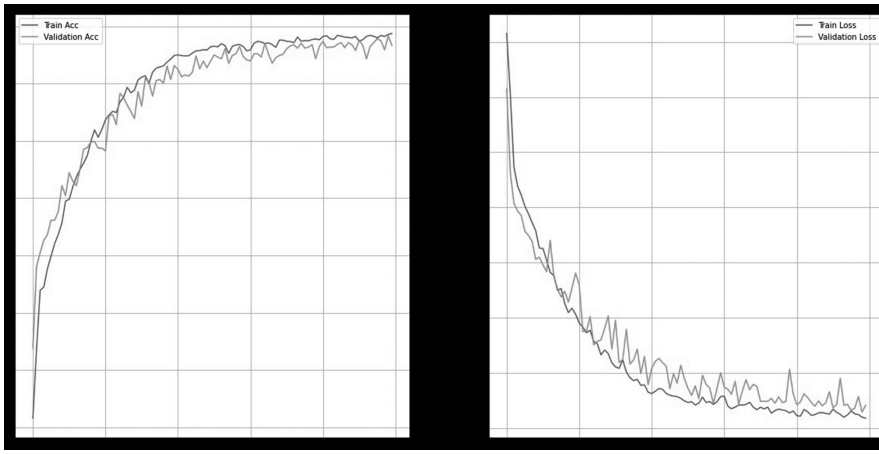


**FIGURE 12.19** (a) The accuracy of training and validation across epochs and (b) the training loss and validation loss across epochs.

**TABLE 12.4**
**Performance of ResNetF**

| CLASS | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| CN | 97% | 0.99 | 0.97 | 0.98 |
| MCI | 97% | 0.99 | 0.97 | 0.98 |
| MD | 100% | 1.00 | 1.00 | 1.00 |
| AD | 99% | 0.95 | 0.99 | 0.97 |
| Avg | 97% | 0.98 | 0.98 | 0.98 |

**TABLE 12.5**
**Confusion Matrix of ResNetF**

| | | Actual Class | | | |
|---|---|---|---|---|---|
| Confusion Matrix | | CN | MCI | MD | AD |
| Predicted class | CN | 618 | 2 | 0 | 20 |
| | MCI | 2 | 173 | 0 | 4 |
| | MD | 0 | 0 | 12 | 0 |
| | AD | 4 | 0 | 0 | 444 |

This chapter aimed to find out whether the early diagnosis of AD can be reliably performed by using MRI of the brain together with a DL algorithm known as a CNN.

Therefore, an enhanced residual neural network known as ResNetF is proposed to classify the four stages of AD by increasing the number of convolutional layers that effectively improve the network's ability to detect as many AD biomarkers as possible. Replacing the activation function (ReLU) with a leaky ReLU can also solve the problem of losing useful features that could help construct high-level discriminative features and reduce training time. To avoid over-fitting, a dropout layer is added before the fully connected layer to train all layers in our architecture from scratch without over-fitting problems.

## REFERENCES


[1] J. Poirier and S. Gauthier, *Alzheimer's Disease: The Complete Introduction*. Dundurn, Canada, 2014.

[2] R. Sahyouni, A. Verma, and J. Chen, *Alzheimer's Disease Decoded: The History, Present, and Future of Alzheimer's Disease and Dementia*. 1st ed. World Scientific, Singapore, 2016.

[3] M. Prince, A. Wimo, M. Guerchet, G. Ali, Y. Wu, M. Prina, *"World Alzheimer Report 2015". The Global Impact of Dementia. An Analysis of Prevalence, Incidence, Cost and Trends,* Alzheimer's Disease International, London, 2015.

[4] J. D. Kelleher, *Deep Learning*. Illustrated edition. The MIT Press, Cambridge, MA, September 10, 2019.

[5] H. Abdulkarim and M. Z. Al-Faiz, "Online multiclass EEG feature extraction and recognition using modified convolutional neural network method," *Int. J. Electr. Comput. Eng.*, vol. 11, no. 5, pp. 4016–4026, 2021, doi: 10.11591/ijece.v11i5.pp4016-4026.

[6] H. Benmeziane, "Comparison of deep learning frameworks and compilers", Thesis for the degree of Master in Computer Science, Université Polytechnique Hauts-de-France, 2020.

[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, doi: 10.1016/j.neunet.2014.09.003.

[8] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, "Deep image: Scaling up image recognition," *arXiv preprint arXiv:1501.02876*, vol. 7, no. 8, 2015.

[9] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[10] M. Wu and L. Chen, "Image recognition based on deep learning," *2015 Chinese Automation Congress (CAC)*, IEEE, pp. 542–546, 2015, doi: 10.1109/CAC.2015.7382560.

[11] Q. Dou et al., "Automatic detection of cerebral microbleeds from mr images via 3D convolutional neural networks," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1182–1195, 2016, doi: 10.1109/TMI.2016.2528129.

[12] U. R. Acharya, H. Fujita, S. L. Oh, Y. Hagiwara, J. H. Tan, and M. Adam, "Application of deep convolutional neural network for automated detection of myocardial infarction using ECG signals," *Inf. Sci. (Ny).*, vol. 415–416, pp. 190–198, 2017, doi: 10.1016/j.ins.2017.06.027.

[13] M. M. Thaha, K. P. M. Kumar, B. S. Murugan, S. Dhanasekeran, P. Vijayakarthick, and A. S. Selvi, "Brain tumor segmentation using convolutional neural networks in MRI images," *J. Med. Syst.*, vol. 43, no. 9, 2019, doi: 10.1007/s10916-019-14160.

[14] M. Rahimzadeh and A. Attar, "A modified deep convolutional neural network for detecting COVID-19 and pneumonia from chest X-ray images based on the concatenation of Xception and ResNet50V2," *Informatics Med. Unlocked*, vol. 19, p. 100360, 2020, doi: 10.1016/j.imu.2020.100360.

[15] Y. Kazemi, "A deep learning pipeline for classifying different stages of Alzheimer's disease from fMRI data", Thesis for the degree of Master in Computer Science, Brock University, 2017.

[16] H. L. J. van der Maas, L. Snoek, and C. E. Stevenson, "How much intelligence is there in artificial intelligence? A 2020 update," *Intelligence*, vol. 87, no. May, p. 101548, 2021, doi: 10.1016/j.intell.2021.101548.

[17] M. Nielsen, *Neural Networks and Deep Learning*. San Francisco, CA: Determination Press, 2015.

[18] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks, " *Int. J. Eng. Appl. Sci. Technol.*, vol. 04, no. 12, pp. 310–316, 2020, doi: 10.33564/ijeast.2020.v04i12.054.

[19] T. Ateeq et al., "Ensemble-classifiers-assisted detection of cerebral microbleeds in brain MRI, " *Comput. Electr. Eng.*, vol. 69, pp. 768–781, Jul. 2018, doi: 10.1016/j.compeleceng.2018.02.021.

[20] M. Al-Smadi, M. Hammad, Q. B. Baker, and S. A. Al-Zboon, "A transfer learning with deep neural network approach for diabetic retinopathy classification, " *Int. J. Electr. Comput. Eng.*, vol. 11, no. 4, pp. 2088–8708, 2021, doi: 10.11591/ijece.v11i4.pp2088-8708.

[21] R. Poojary, R. Raina, and A. K. Mondal, "Effect of data-augmentation on fine-tuned cnn model performance, " *IAES Int. J. Artif. Intell.*, vol. 10, no. 1, pp. 84–92, 2021, doi: 10.11591/ijai.v10.i1.pp84-92.

[22] S. B. Jadhav, V. R. Udupi, and S. B. Patil, "Convolutional neural networks for leaf image-based plant disease classification, " *IAES Int. J. Artif. Intell.*, vol. 8, no. 4, pp. 328–341, 2019, doi: 10.11591/ijai.v8.i4.pp328-341.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Illustrated edition. The MIT Press, Cambridge, MA, November 18, 2016.

[24] R. Yamashita, M. Nishio, R. Do and K. Togashi, "Convolutional neural networks: An overview and application in radiology, " *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, 2018, doi: 10.1007/s13244-018-0639-9.

[25] G. Currie, "Intelligent imaging: Anatomy of machine learning and deep learning, " *J. Nucl. Med. Technol.*, vol. 47, no. 4, pp. 273–281, 2019, doi: 10.2967/JNMT.119.232470.

[26] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks, " *2017 International Conference on Communication and Signal Processing (ICCSP)*, IEEE, pp. 588–592, 2017, doi: 10.1109/ICCSP.2017.8286426.

[27] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks, " *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020, doi: 10.1007/s10462-020-09825-6.

[28] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on MRI, " *Z. Med. Phys.*, vol. 29, no. 2, pp. 102–127, 2019, doi: 10.1016/j.zemedi.2018.11.002.

[29] P. Mianjy and R. Arora, "On convergence and generalization of dropout training, " *Adv. Neural Inf. Processing Syst.*, vol. 33, pp. 21151–21161, 2020.

[30] A. Labach, H. Salehinejad, and S. Valaee, "Survey of dropout methods for deep neural networks, " *arXiv preprint arXiv:1904.13310*, 2019.

[31] Q. Xu, M. Zhang, Z. Gu, and G. Pan, "Overfitting remedy by sparsifying regularization on fully-connected layers of CNNs, " *Neurocomputing*, vol. 328, pp. 69–74, 2019, doi: 10.1016/j.neucom.2018.03.080.

[32] M. S. AL-Huseiny and A. S. Sajit, "Transfer learning with GoogLeNet for detection of lung cancer, " *Indones. J. Electr. Eng. Comput. Sci.*, vol. 22, no. 2, pp. 1078–1086, 2021, doi: 10.11591/ijeecs.v22.i2.pp1078-1086.

[33] J. Heaton, *Artificial Intelligence for Humans, Volume 3: Neural Networks and Deep Learning.* Heaton Research, Inc., CreateSpace Independent Publishing Platform, October 28, 2015.

[34] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, San Francisco, CA, 2015.

[35] R. A. Minhas, A. Javed, A. Irtaza, M. T. Mahmood, and Y. B. Joo, "Shot classification of field sports videos using AlexNet convolutional neural network, " *Appl. Sci.*, vol. 9, no. 3, 2019, doi: 10.3390/app9030483.

[36] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti and D. De, "Fundamental concepts of convolutional neural network", In *Recent trends and advances in artificial intelligence and Internet of Things 2020* (pp. 519–567). Springer, Cham.

[37] M. Mishra, T. Choudhury and T. Sarkar, "CNN based efficient image classification system for smartphone device ", 2021, doi: 10.21203/rs.3.rs-428430/v1.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[39] P. C. Nissimagoudar, A. V. Nandi, A. Patil, and H. M. Gireesha, "AlertNet: Deep convolutional-recurrent neural network model for driving alertness detection, " *Int. J. Electr. Comput. Eng.*, vol. 11, no. 4, pp. 2088–8708, 2021, doi: 10.11591/ijece.v11i4.pp2088-8708.

[40] M. A. Ihsan Aquil and W. H. Wan Ishak, "Evaluation of scratch and pre-trained convolutional neural networks for the classification of tomato plant diseases, " *IAES Int. J. Artif. Intell.*, vol. 10, no. 2, pp. 467–475, 2021, doi: 10.11591/ijai.v10.i2.pp467-475.

[41] E. M. Benyoussef, A. Elbyed, and H. El Hadiri, "Data mining approaches for Alzheimer's disease diagnosis," *International Symposium on Ubiquitous Networking.* Springer, Cham, vol. 10542 LNCS, pp. 619–631, 2017, doi: 10.1007/978-3-319-68179-5_54.

[42] M. Liu et al., "A multi-model deep convolutional neural network for automatic hippocampus segmentation and classification in Alzheimer's disease", *NeuroImage*, vol. 208, p. 116459, 2020.

[43] R. Jain, N. Jain, A. Aggarwal, and D. J. Hemanth, "Convolutional neural network based Alzheimer's disease classification from magnetic resonance brain images," *Cogn. Syst. Res.*, vol. 57, pp. 147–159, 2019, doi: 10.1016/j.cogsys.2018.12.015.